Component-based Modeling of Real-Time Systems

ARTIST2 / UNU-IIST Spring School in Xian, China on Models, Methods and Tools for Embedded Systems
April 3rd-15th, 2006

Joseph Sifakis
VERIMAG
sifakis@imag.fr

In collaboration with
Gregor Goessler (INRIA)
Ananda Basu, Marius Bozga, Susanne Graf (Verimag)
Modeling plays a central role in systems engineering
• Can profitably replace experimentation on actual systems
• Can provide a basis for rigorous system development and implementation (model-based approaches).

Modeling real-time systems
• Raises hard problems about concepts, languages and their semantics e.g. What is an architecture? What is a scheduler? How synchronous and asynchronous systems are related?
• Requires a deep understanding of basic system design issues such as development methodologies (combination of techniques and tools, refinement) and architecture design principles.
Move from physical prototypes to virtual prototypes (models) with obvious advantages: minimize costs, flexibility, genericity, formal validation is a possibility.

We need modeling and validation environments for complex real-time systems:

- Libraries of Components
  - ex. HW, SW, Models of continuous dynamic systems
- Languages and tools for assembling components

Synthesize embedded software from domain-specific models
- ex. Matlab, SystemC, UML, SDL.
Modeling real-time systems – research objectives

Develop a rigorous and general basis for architecture modeling and implementation:

• Study the concept of architecture as a means to organize computation (behavior, interaction, control)

• Define a meta-model for real-time architectures, encompassing specific styles, paradigms, e.g. modeling
  - Synchronous and asynchronous execution
  - Event driven and data driven interaction
  - Distributed execution
  - Architecture styles such as client-server, blackboard architecture

• Provide automated support for component integration and generation of glue code meeting given requirements
Existing approaches involving components

- Architecture Description Languages focusing on non-functional aspects e.g. ADL, AADL or SW Design Languages
- Modeling languages: Statecharts, UML, Simulink/Stateflow, Metropolis, Ptolemy
- Coordination languages extensions of programming languages: Linda, Javaspaces, TSpaces, Concurrent Fortran, SystemC, NesC
- Middleware standards e.g. IDL, Corba, Javabeans, .NET
- Software development environments: PCTE, SWbus, Softbench, Eclipse
- Process algebras and automata e.g. Pi-Calculus
Overview – Part 1

- Modeling real-time systems
  - The problem
  - Heterogeneity
  - Component-based construction

- Interaction modeling
  - Definition
  - Composition
  - Deadlock-freedom preservation

- Priority modeling
  - Definition
  - Composition

- The BIP framework
  - Implementation
  - Timed components
  - Event triggered vs. Data triggered

- Discussion
Overview – Part 2

• Timed systems
  – Definition
  – Examples

• Scheduler modeling
  – The role of schedulers
  – Control invariants
  – Scheduler specifications
  – Composability results

• Timed systems with priorities
  – Definition
  – Composition of priorities
  – Correctness-by-construction results

• The IF toolset
A Timed Model of a RT system can be obtained by “composing” its application SW with timing constraints induced by both its execution and its external environment.
## Modeling real-time systems - our approach

<table>
<thead>
<tr>
<th></th>
<th>Application SW</th>
<th>Timed model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DESCRIPTION</strong></td>
<td>Reactive machine (untimed)</td>
<td>Reactive machine + External Environment + Execution Platform</td>
</tr>
<tr>
<td><strong>TIME</strong></td>
<td>Reference to physical (external) time</td>
<td>Quantitative (internal) time Consistency pbs - timelocks</td>
</tr>
<tr>
<td><strong>TRIGGERING</strong></td>
<td>Timeouts to control waiting times</td>
<td>Timing constraints on interactions</td>
</tr>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
<td></td>
</tr>
<tr>
<td><strong>ACTIONS</strong></td>
<td>No assumption about Execution Times Platform-independence</td>
<td>Assumptions about Execution Times Platform-dependence</td>
</tr>
</tbody>
</table>
Modeling real-time systems - our approach

Application SW
Platform Timed Model
Environment Timed Model
User Requirements

Composition/Synthesis

Code Generation
System Timed Model
Analysis

Implementation
Diagnostics

Component-based modeling
Modeling real-time systems – Taxys (1)

Environment

Deadline constraint:
\[ t_{\text{out}} - t_{\text{in}} < D \]

Throughput constraint:
no buffer overflow
Modeling real-time systems – Taxys (2)

- ESTEREL + C Data
- SAXO-RT
- C Code
- SAXO
- Machine Description
- Target Machine executable code
- Event Handler Timed Model
- Environment Timed Model
- Timed (instrumented) C Code
- C2TimedC
- Exec. Times
- IF/KRONOS
- Timing Diagnostics
Modeling real-time systems – Taxys(3)

Application = ESTEREL + Pragmas

Event Handler

Environment = ESTEREL + Pragmas

Exec.Times

QoS requ.

SAXO-RT

Instrumented C Code

KRONOS Algorithms and Data Structures

Target Machine Executable Code

SAXO

IF/KRONOS

Timing Diagnostics

Instrumented C Code
Overview – Part 1

• Modeling real-time systems
  – The problem
  – Heterogeneity
  – Component-based construction

• Interaction modeling
  – Definition
  – Composition
  – Deadlock-freedom preservation

• Priority modeling
  – Definition
  – Composition

• The BIP framework
  – Implementation
  – Timed components
  – Event triggered vs. Data triggered

• Discussion
Heterogeneity – Abstraction Levels

- Model (requirements)
- Application Software
- System
- Execution Platform
Heterogeneity - from application SW to implementation

Application SW

Matlab/Simulink
Lustre
Esterel
C
C++
RT- Java
UML

Implementation

DSP µcontroller
TTA
CAN
RTOS
CORBA
OSEK
Heterogeneity - from application SW to implementation

Functional properties - logical abstract time
High level structuring constructs and primitives
Simplifying synchrony assumptions wrt environment

Non functional properties, involving time and quantities
Task coordination, scheduling, resource management,
Execution times, interaction delays, latency

Application SW
Heterogeneity - synchronous vs. asynchronous execution

**Synchronous**
- Lustre, Esterel
- Statecharts
- Non interruptible execution steps
- Usually, single task, single processor
- «Everybody gets something »

**Asynchronous**
- ADA, SDL
- Event triggered
- Multi-tasking - RTOS
- Usually, static Priorities
- «Winner takes all »

*Application SW*

*Component-based approaches*

*Implementation*
Heterogeneity - interaction

Interactions can be
- strict (CSP) or non strict (SDL, Esterel)
- atomic (CSP, Esterel) or non atomic (SDL)
- binary (point to point as in CCS, SDL) or n-ary in general
Heterogeneity - example

A: Atomic interaction

B: Blocking interaction

Asynchronous Computation

Synchronous Computation

Java
UML

Lotos
CSP

A B
nonA B
A nonB
nonA nonB

SDL
UML

Matlab/Simulink
VHDL/SystemC
Statecharts

A: Atomic interaction

B: Blocking interaction
Overview – Part 1

• Modeling real-time systems
  – The problem
  – Heterogeneity
  – Component-based construction

• Interaction modeling
  – Definition
  – Composition
  – Deadlock-freedom preservation

• Priority modeling
  – Definition
  – Composition

• The BIP framework
  – Implementation
  – Timed components
  – Event triggered vs. Data triggered

• Discussion
Component-based construction – atomic components

Build systems by **composition of atomic components**

**Atomic components** are building blocks composed of **behavior** and **interface**

- **Behavior** is represented by a transition system
- **Interface** hides irrelevant internal behavior and provides some adequate abstraction for composition and re-use, e.g. set of ports (action names) and associated variables
Component-based construction – formal framework

Pb: Build a component \( C \) satisfying a given property \( P \), from
- \( C_0 \) a set of atomic components
- \( \mathcal{G} = \{ gl_1, \ldots, gl_i, \ldots \} \) a set of glue operators on components

Glue operators
- model mechanisms used for communication and control such as protocols, schedulers, buses
- restrict the behavior of their arguments, that is the projection of the behavior of \( gl(C_1, C_2, \ldots, C_n) \) on actions of \( C_1 \) is contained in the behavior of \( C_1 \)
Component-based construction – formal framework

Operational Semantics: the meaning of a compound component is an atomic component

Algebraic framework:
- Components are terms of an algebra of terms \((\mathcal{C}, \simeq)\) generated from \(\mathcal{C}_0\) by using operators from \(\mathcal{G}\)
- \(\simeq\) is a congruence compatible with operational semantics
Component-based construction - requirements

Examples of existing frameworks:
• Sequential functions with logical operators and delay operators for building circuits
• Process algebras
• Distributed algorithms define generic glue operators for a given property $P$ e.g. token ring, clock synchronization

Pb: Find a set of glue operators meeting the following requirements:
• Incremental description
• Correctness-by-construction
• Expressiveness (discussed later)
Flattening can be achieved by introducing an idempotent operation $\oplus$ such that $(GL, \oplus)$ is a commutative monoid and $gl(gl'(C_1, C_2, \ldots, C_n)) \cong gl \oplus gl'(C_1, C_2, \ldots, C_n)$.
Component-based construction - Correctness by construction: compositionality

Build correct systems from correct components

\[ c_i \text{ sat } P_i \text{ implies } \forall gl \exists \tilde{gl} \]  
\[ \exists c_1 \ldots c_n \text{ sat } \tilde{gl}(P_1, \ldots, P_n) \]

We need compositionality results about preservation of progress properties such as deadlock-freedom and liveness.
Component-based construction - Correctness by construction: composability

Make the new without breaking the old

Property stability phenomena are poorly understood
- feature interaction
- non composability of scheduling algorithms
Component-based construction - compositionality vs. composability
Overview – Part 1

• Modeling real-time systems
  – The problem
  – Heterogeneity
  – Component-based construction

• Interaction modeling
  – Definition
  – Composition
  – Deadlock-freedom preservation

• Priority modeling
  – Definition
  – Composition

• The BIP framework
  – Implementation
  – Timed components
  – Event triggered vs. Data triggered

• Discussion
Component-based construction – The BIP framework

Layered component model

Priorities (Memoryless Controller)

Interaction Model (Connectors on typed ports)

Composition (incremental description)
An atomic component has

- A set of ports $P$
- A set of control locations $S$
- A set of variables $V$
- A set of transitions of the form
  - $p$ is a port
  - $g_p$ is a guard, boolean expression on $V$
  - $f_p$ is a function on $V$ (block of code)
Atomic components – behavior

p: a port through which interaction is sought

$g_p$: a pre-condition for interaction through p

$f_p$: a computation (local state transformation)

**Semantics**: interaction followed by computation
- A transition is enabled if $g_p$ is true and some interaction involving p is possible
- The execution of the enabled transition involves the execution of an interaction involving p followed by the execution of $f_p$
Interaction modeling

- A **connector** is a set of ports which can be involved in an interaction.

- Port types (complete ▼, incomplete ○) are used to distinguish between ports which **may** or **must** interact.

- An **interaction** of a connector is a non-empty subset of its set of ports such that: either it contains some complete port or it is maximal.

Interactions:
{tick1, tick2, tick3} {out1} {out1, in2} {out1, in3} {out1, in2, in3}
Interaction modeling – connectors

- cl1, cl2
- out, in
- in1, out, in2

Diagram showing relationships between different components and their interactions.
Interaction modeling - composition

CN\([K_1,K_2]\): \{p_1, p_2, p_3, p_4\}, \{p_{11}, p_{12}\}

CN\([K_1]\): \{p_1, p_2\}, \{p_5, p_9\}, \{p_6, p_9\}

CN\([K_2]\): \{p_3, p_4\}, \{p_7, p_{10}\}, \{p_8, p_{10}\}
Interaction modeling – composition (2)

$$CN[K_1,K_2] = \text{max } CN[K_1] \cup CN[K_2] \cup CN[K_1,K_2]$$

$CN[K_1]: \{p_1, p_2, p_5, p_9, p_6, p_9\}$

$CN[K_2]: \{p_3, p_4, p_7, p_{10}, p_8, p_{10}\}$

$K_1 \cup K_2$
Interaction modeling – results [Goessler Sifakis 2003]

Incremental commutative composition encompassing blocking and non blocking interaction
Interaction modeling - composition: operational semantics

CN: {put, get}
Interaction modeling - connector semantics: data transfer

**CN:** BUS={send, rec1, rec2}

- {send}: true $\rightarrow$ skip
- {send, rec1}: $x < y \rightarrow x := y - x$, $y := y + x$
- {send, rec2}: $x < z \rightarrow x := z - x$, $z := z + x$
- {send, rec1, rec2}: $x < z + y \rightarrow x := y + z - x$, $y := y + x$, $z := z + x$

**Maximal progress:** execute a maximal enabled interaction
Interaction modeling - composition: operational semantics

CN: BUS={p₁, p₂, ..., pₙ, ..., pₛ}
............................
π = {p₁, p₂, ..., pₙ}: Gₚ → Fₚ
............................

\[ g_π = g_{p₁} \land g_{p₂} \land ... \land g_{pₙ} \land G_π \]

\[ f_π = F_π; f_{p₁}, f_{p₂}, ..., f_{pₙ} \]

Maximal progress: execute a maximal enabled interaction
Interaction modeling – example: Producer-Consumer

Producer

Resource

Consumer

put(item, buffer)

Item:=first(buffer)

Item:=first(buffer)
The BIP framework – Event Triggered Mod8 counter
Interaction models - commitment protocol

\[ \text{CN} : \{\text{vote}\} \cup \{\text{vote}_i\}_{i \in I}, \{\text{commit}\} \cup \{\text{commit}_i\}_{i \in I}, \{\text{yes}\} \cup \{\text{yes}_i\}_{i \in I} \]

\[ \text{CI} : \text{abort}, \text{no}, \text{no}_i \text{ for } i \in I \]
Interaction models - commitment protocol (2)

\[ CN : \{\text{vote}\} \cup \{\text{vote}_i\}_{i \in I}, \{\text{commit}\} \cup \{\text{commit}_i\}_{i \in I}, \{\text{yes}, \text{yes}_i\}_{i \in I} \text{ for } i \in I \]

\[ CI: \text{abort, no, no}_i, \text{abort}_i \text{ for } i \in I \]
Interaction models - checking for deadlock-freedom

For a given system (set of components + interaction model), its dependency graph is a bipartite labeled graph with

Nodes $N = \text{Set of components} \cup \text{Set of minimal interactions}$

Edges $E$
- $(\alpha, a, k) \in E$ if $\alpha$ is an interaction, $a \in \alpha$ is an incomplete action of $k$
- $(k_1, a_1, \alpha) \in E$ if $a_1 \in \alpha$ is an action of $k_1$

Blocking condition for an incomplete action $a$:
$$Bl(a) = en(a) \land \neg (en(a_1) \land en(a_2) \land en(a_3))$$
Theorem 1: A system is deadlock-free if its atomic components have no deadlocks and its dependency graph has a backward closed subgraph such that for all its circuits $\omega$

$$\text{Bl}(\omega) = \bigwedge_{a \in \omega} \text{Inc}(\omega) \land \text{Bl}(a) = false$$

where $\text{Inc}(\omega) = \bigwedge_{k \in \omega} \text{Inc}(k)$ with $\text{Inc}(k)$ the set of the states of $k$ from which only incomplete actions can be executed.
Interaction models - checking for deadlock-freedom: example

- **CN**: \{put, get₁, get₂\}
- **MCI**: \{put, get₁\}, \{put, get₂\}

Diagram:
- **Producer**
- **Consumer₁**
- **Consumer₂**
- **Put**
Interaction models - checking for deadlock-freedom: example

\[ \omega_1 = (\text{producer}, n_1, \text{consumer}_2, n_2) \]
\[ \text{Bl}(\omega_1) = \text{false} \]

\[ \omega_2 = (\text{producer}, n_2, \text{consumer}_1, n_1) \]
\[ \text{Bl}(\omega_2) = \text{false} \]

\[ \omega_3 = (\text{consumer}_1, n_1, \text{consumer}_2, n_2) \]
\[ \text{Bl}(\omega_3) = \text{Inc}(\omega_3) \land \text{en}(\text{get}_1) \land \neg (\text{en}(\text{get}_2) \land \text{en}(\text{put})) \]
\[ \land \text{en}(\text{get}_2) \land \neg (\text{en}(\text{get}_1) \land \text{en}(\text{put})) \]
\[ = \text{Inc}(\omega_3) \land \text{en}(\text{get}_1) \land \text{en}(\text{get}_2) \land \neg \text{en}(\text{put}) \]

Deadlock-freedom if \( \text{Inc}(\text{producer}) \land \neg \text{en}(\text{put}) = \text{false} \)
Interaction models - checking for individual deadlock-freedom

**Definition:** A component of a system is individually deadlock-free if it can always perform some action.

**Theorem 2:** Sufficient condition for individual deadlock-freedom of a component $k$

- $k$ belongs to a backward closed subgraph of a dependency graph satisfying conditions of Theorem 1;
- In any circuit of this subgraph, all its components are controllable with respect to their outputs i.e. it is always possible by executing complete interactions, to reach states enabling all the output actions of the component;
- All the $n$-ary interactions for $n>2$ are strong synchronizations.

Gregor Goessler and Joseph Sifakis "Component-based construction of deadlock-free systems"  
Interaction models - discussion

- The distinction **interaction model / behavior** is crucial or the model construction methodology. Layered description => separation of concerns => associativity

- Different from other approaches e.g. process calculi, which combine behavior composition operators and restriction/hiding operators at the same level.

\[(\text{P1} \parallel \text{P2}) \backslash a \parallel \text{P3}) \backslash a' \rightarrow \backslash a \oplus \backslash a' \rightarrow \text{P1} \parallel \text{P2} \parallel \text{P3}\]

- Framework encompassing strong and weak synchronization
Overview – Part 1

- Modeling real-time systems
  - The problem
  - Heterogeneity
  - Component-based construction

- Interaction modeling
  - Definition
  - Composition
  - Deadlock-freedom preservation

- Priority modeling
  - Definition
  - Composition

- The BIP framework
  - Implementation
  - Timed components
  - Event triggered vs. Data triggered

- Discussion
Priority modeling

Restrict non-determinism by using (dynamic) priority rules

Precedence rule: 

\[ \text{true} \rightarrow p_1 < p_2 \]  
\[ C \rightarrow p_1 < p_2 \]  

\[ g_1' = g_1 \land \neg g_2 \]

\[ g_1' = g_1 \land \neg (C \land g_2) \]
A priority order is a strict partial order \( \preceq \subseteq \text{Inter} \times \text{Inter} \)

A set of priority rules, \( \text{PR} = \{ C_i \rightarrow \langle \rangle_i \} \) where \( \{C_i\}_i \) is a set of disjoint state predicates

\[
g'_k = g_k \land \bigwedge C \rightarrow \langle \in \text{PR} \ (C \Rightarrow \bigwedge_{pk} \langle pi \rightarrow g_j \rangle) \]
Priority modeling - FIFO policy

PR : $t_1 \leq t_2 \rightarrow b_1 \langle b_2$ 
$t_2 \leq t_1 \rightarrow b_2 \langle b_1$
Priority modeling - EDF policy

PR: \( D_1 - t_1 \leq D_2 - t_2 \rightarrow b_2 \prec b_1 \)

\( D_2 - t_2 \leq D_1 - t_1 \rightarrow b_1 \prec b_2 \)
Priority modeling - Composition

PR2
PR1 ≠
PR1
PR2

a \langle 1 \ b
b \langle 2 \ c
a \langle 1 \ b
b \langle 2 \ c
a \langle 1 \ b
b \langle 2 \ c
a \langle 1 \ b
b \langle 2 \ c
Priority modeling– Composition (2)

We take:

\[ PR2 \oplus PR1 \]

\[ \Delta \]

\[ PR1 \oplus PR2 \]

PR1\(\oplus\) PR2 is the least priority containing PR1\(\cup\)PR2

Results:

- The operation \(\oplus\) is partial, associative and commutative
- \(PR1(PR2(B)) \neq PR2(PR1(B))\)
- \(PR1 \oplus PR2(B) \text{ refines } PR1 \cup PR2(B) \text{ refines } PR1(PR2(B))\)
- Priorities preserve deadlock-freedom
Priority modeling - mutual exclusion + FIFO policy

\[
\begin{align*}
\text{t1} & \leq \text{t2} \rightarrow b1 \prec b2 \\
\text{t2} & \leq \text{t1} \rightarrow b2 \prec b1 \\
\text{true} & \rightarrow b1 \prec f2 \\
\text{true} & \rightarrow b2 \prec f1
\end{align*}
\]
Priority modeling– mutual exclusion: example

Risk of deadlock: $PR \oplus PR'$ is not defined
Priority modeling – run to completion

\[ i_1 \preceq \{o_1,i_2\} \preceq \{o_2,i_3\} \preceq o_3 \]

CN: \{o_1,i_2\}, \{o_2,i_3\}
Overview – Part 1

- Modeling real-time systems
  - The problem
  - Heterogeneity
  - Component-based construction

- Interaction modeling
  - Definition
  - Composition
  - Deadlock-freedom preservation

- Priority modeling
  - Definition
  - Composition

- The BIP framework
  - Implementation
  - Timed components
  - Event triggered vs. Data triggered

- Discussion
The BIP framework – related approaches

Vanderbilt’s Approach

- Semantic Unit Meta-model
- Composition Operators
- Behavior
- Operational Semantics
- ASML
- .net

Metropolis

- Semantic Domain
- Quantity Manager
- Media
- Behavior
- Operational Semantics
- Platform

PTOLEMY

- MoC (Model of Computation)
- Director
- Channels
- Behavior
- Operational Semantics
- Platform
A system is defined as a point of the 3-dimensional space. Separation of concerns: any combination of coordinates defines a system.
The BIP framework – model construction space

Non separation of concerns for PTOLEMY
The BIP framework – property preservation

- Deadlock-free
- Invariant

Architecture
System
pr
im
+restriction
+interaction
+refinement

B
The BIP framework – classes of components

Characterize relations between classes by elementary model transformations:
- Untimed-timed
- Synchronous – asynchronous
- Event triggered – data triggered
Implementation - work at Verimag

- Graphic language: AADL or UML
- BIP
- C++
- THINK
- BIP Platform
- IF Platform
- IF
Implementation - generation of C++ code from BIP

```
C -> a\langle b
BIP model
```

Component Meta-model

Interaction Meta-model

Dynamic priorities Meta-model

Execution Engine

PLATFORM
Implementation - The execution platform
Implementation - The execution platform: the engine

1. **Init**
   - Launch atom's threads

2. **Loop**
   - Wait all atoms
   - Compute legal interactions
   - Choose among maximal
   - Choose

3. **Execute**
   - Notify involved atoms
   - Execute chosen interaction transfer

4. **Filter**
   - Filter w.r.t. priorities

5. **Ready**

6. **Stable**

Diagrams provide a visual representation of the execution process, highlighting the sequence of events and the interactions between different stages.
Implementation - BIP atomic component: abstract syntax

component C
port complete: p1, … ; incomplete: p2, …
data {# int x, float y, bool z, …. #}
init {# z=false; #}

behavior
state s1
               on p1 provided g1 do f1 to  s1’
               ..................      ......  
               on pn provided gn do fn to  sn’

state s2
       on ..... 
       ....

state sn
       on ....

end
end
Implementation - BIP connectors and priorities

**connector** BUS= \{p, p', ... , \}
complete()
behavior
  on $\alpha_1$ provided $g_{\alpha_1}$ do $f_{\alpha_1}$
  on $\alpha_2$ provided $g_{\alpha_2}$ do $f_{\alpha_2}$
end

**priority** PR
  if C1 ($\alpha_1 < \alpha_2$), ($\alpha_3 < \alpha_4$), ...
  if C2 ($\alpha < ...$), ($\alpha < ...$), ...
  ...
  if Cn ($\alpha < ...$), ($\alpha < ...$), ...
Implementation - BIP compound component

component name
contains c_name1 i_name1(par_list)
 ...... 
contains c_namen i_namen(par_list)

connector name1
 ...... 
connector namem

priority name1
 ...... 
priority namek
end
run() {
    Port* p;
    int state = 1;
    while(true) {
        switch(state) {
            case 1:
                p = sync(a, g_a, d, g_d);
                if (p == a)
                    f_a;  state = 2;
                else
                    f_d;  state = 3;
                break;
            case 2:
                p = sync(b, g_b, e, g_e);
                ...
            case 3: ...
        }
    }
}
Timed components

PR: red_guards → tick \ all_other_ports
Timed components: Preemptable task

- **IDLE**
  - **tick** delay++
  - time++

- **READY**
  - **arrive**, time = P
    - time := 0
  - **tick** delay++
    - time++

- **EXEC**
  - **start**, time <= P - WCET
    - delay := 0
  - **tick** delay++
    - time++

- **SUSPEND**
  - **preempt**
  - **tick** time++

- **IDLE** to **READY** transition on **arrive** event
- **READY** to **EXEC** transition on **start** event
- **EXEC** to **SUSPEND** transition on **preempt** event
- **SUSPEND** to **EXEC** transition on **resume** event
Timed components - Case study : Problem 1

Bursty Event-Stream:
- Period = 10
- Jitter = 50
- Min. Interarrival Dist. = 1

CPU1
  **T1**
  WCED = 8

CPU2
  **T2**
  WCED = 4

CPU3
  **T3**
  WCED = 1
Timed components - Case study : Problem 1

Component Task:

**Task**

- **READY**
  - get count++
  - tick

- **EXEC**
  - tick delay++
  - get count++
  - start, (count>0)
  - count--, delay:=0

- **finish**
  - finish, [delay<= WCET ]
Timed components - Case study : Problem 1
BIP code snippet for Task

component Task (int wcet)
  port get, start, tick, finish
  data {# int count, delay; #}
  ...
  init {# count = 0;
    WCET = wcet;
    ...
    #}
  behavior
    state READY
      on get do {# count++; #} to READY
      on start provided {# count > 0 #} do {# count--; delay = 0; #} to EXEC
      on tick to READY
    state EXEC
      on get do {# count++; #} to EXEC
      on finish when ({# delay <= WCET #}, delayable) to READY
      on tick do {# delay++; #} to EXEC
    end
  ...
end
Timed components - Bursty Event Stream Generator

Bursty Event-Stream for
Period = T
Jitter = J
Min. Interarrival Dist. = d

\[ x := 0 \]
\[ y := 0 \]
\[ k := 0 \]

\[ a_{i-1} \quad a_i \]

\[ i \quad i+1 \quad \ldots \quad i+k \]

\[ x \quad y \]

\[ [k \geq 0 \land x + k \cdot T \leq J \land y \geq d] \delta \]
\[ y := 0, \; k := k - 1 \]
Timed components - Case study: Problem 1
Composition in BIP glue

PR: tick \{ EvntT1, T1T2, T2T3, T3.Finish \}
Timed components - Case study : Problem 1
BIP code snippet for Task Composition

class System
contains Launcher eventGenerator(10, 5, 1)
contains Task T1(8), T2(4), T3(1)

connector Tick = eventGenerator.tick, T1.tick, T2.tick, T3.tick
behavior
end
connector EvntT1 = eventGenerator.go, T1.get
behavior
end
...
priority // start < get ( no event losses )
getStart1 T1.Start : T1.start < EvntT1 : T1.get
...
priority // finish < get ( no event losses )
getFin1 T1T2 : T1.finish < EvntT1 : T1.get
...
priority // tick < get_i ( => tick < finish_i-1 )
getTick2 if (T1.delay == T1.WCET) Tick : T2.tick < T1T2 : T2.get
...

Timed components - Case study: Problem 2

Bursty Event-Stream:
- Period = 10
- Jitter = 50
- Min. Interarrival Dist. = 1

Preemptive Fixed-Priority Scheduling:
- (T1 has higher priority than T3)

CPU1
- T1
- WCED = 8

CPU2
- T2
- WCED = 4

End-to-end Delay?
- T3
- WCED = 1
Timed components - Case study: Problem 2
Behavior & Architecture def
Timed components - Case study : Problem 2
Composition in BIP glue

PR:
{ T1.preempt, T3.start} < T1.finish
T3.start < T1.start (static priority)
### Timed components – Case study: Max End-to-End Delays

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Non-preemptive</td>
<td>Preemptive</td>
</tr>
<tr>
<td>J=30, P=10, d=1</td>
<td>35</td>
<td>48</td>
<td>40</td>
</tr>
<tr>
<td>J=40, P=10, d=1</td>
<td>43</td>
<td>57</td>
<td>49</td>
</tr>
<tr>
<td>J=50, P=10, d=1</td>
<td>51</td>
<td>66</td>
<td>58</td>
</tr>
</tbody>
</table>
Timed components – MPEG encoder
Timed components – MPEG encoder (2)
Timed components - Billiards
Timed components - Billiards: Co-ordinate component

Coordinate X

\[
\begin{align*}
\text{POS:} & \quad \text{flip} \quad x = 0 \quad v_x := -v_x \\
\text{NEG:} & \quad \text{flip} \quad x = \text{MAX} \quad v_x := -v_x
\end{align*}
\]

\[
\begin{align*}
\text{tick:} & \quad x > 0 \quad x := x + v_x \\
\text{shock:} & \quad \text{other conditions}
\end{align*}
\]
Timed components - Billiards: Ball(Compound)

Coordinate X

- **NEG**
  - x = MAX
  - vx := -vx

- **POS**
  - x < MAX
  - x := x + vx

- **Tick**
  - x > 0
  - x := x + vx

- **Shake**

- **Tick**
  - x = 0
  - vx := -vx

Coordinate Y

- **NEG**
  - y = MAX
  - vy := -vy

- **POS**
  - y < MAX
  - y := y + vy

- **Tick**
  - y > 0
  - y := y + vy

- **Shake**

- **Tick**
  - y = 0
  - vy := -vy

A Ball
Timed components - Billiards: the model

PR: tick $\langle \{\text{shock, flip}\} \rangle$

CN: shock

$g_{\text{shock}}: y_1 = y_2 \land x_1 = x_2$

$f_{\text{shock}}: v_{x_1}, v_{x_2}, v_{y_1}, v_{y_2} := -v_{x_2}, -v_{x_1}, -v_{y_2}, -v_{y_1}$

CN: tick

BALL_1

BALL_2
Timed components: Preemptable task

1. **Start**
   - **Tick**: time++
   - **Delay**: delay++

2. **IDLE**
   - **Arrival, time = P**
     - time:=0
   - **Tick**: delay++
     - time++

3. **READY**
   - **Tick**: delay++
     - time++
   - **Start, time <= P-WCET**
     - delay:=0

4. **EXEC**
   - **Tick**: delay++
     - time++

5. **SUSPEND**
   - **Tick**: time++

6. **Resume**
   - **Resume**

**Preempt**

**Arrive**

**Finish**

**Delay**

**Time**

**Tick**

**Delay**

**Time**
Timed components: fixed priority preemptive scheduling

PR1 (priority for access to the resource):
For $n \geq i > j \geq 1 \{ \text{start}_i, \text{resume}_i \} \prec \{ \text{start}_j, \text{resume}_j \}$

PR2 (non pre-emption by lower pty tasks):
For $n \geq i > j \geq 1 \text{start}_i \text{preempt}_j \prec f_j, \text{resume}_i \text{preempt}_j \prec f_j$

PR3: minimal priority for tick wrt eager guards

CN: For $n \geq i, j \geq 1 \{ \text{start}_i, \text{preempt}_j \} \{ \text{resume}_i, \text{preempt}_j \}$
Timed components: fixed priority preemptive scheduling

PR1 (priority for access to the resource):
For $n \geq i > j \geq 1$ \{\text{start}_i, \text{resume}_i\} \prec \{\text{start}_j, \text{resume}_j\}$

PR2 (non-pre-emption by lower pty tasks):
For $n \geq i > j \geq 1$ \text{preempt}_i \prec \text{finish}_j$, if \text{ready}_i or \text{suspend}_i

PR3: minimal priority for tick wrt eager guards

PR_mutex: \{\text{start}_i, \text{resume}_i\} \prec \{\text{finish}_j, \text{preempt}_j\}$ for $n \geq i, j \geq 1$
Event triggered vs. Data triggered

Decoupling interaction from internal computation:

Event-triggered

Data-triggered
Event Triggered vs. Data Triggered

PR: i1,o1,i2,o2 \langle \tau \rangle
From Data Triggered to Synchronous

**PR: a, b, i1, o3 \(\tau\)**

**CN: a={o1, i2}, \(f_a: x2:=y1\)**

\[ y1:=f(x1) \]

**CN: b={o2, i3}, \(f_b: x3:=y2\)**

\[ y2:=g(x2) \]

\[ y3:=h(x3) \]

---

**PR: s \(\tau\)**

**CN: s= \{s1, s2, s3\}, \(g_s: true, f_s: x2:=y1, x3:=y2\)**

\[ y1:=f(x1) \]

\[ y2:=g(x2) \]

\[ y3:=h(x3) \]
Flip-flop: Event Triggered model

\[ z = \neg x \land \neg y \]

states transition diagram:

- (x=0) → (y=0)
  - px: 0 → px: 1
  - py: 0 → py: 1
  - pz: 0 → pz: 1

- (x=0) → (y=0)
  - px: 1 → px: 0
  - py: 1 → py: 0
  - pz: 1 → pz: 0

- (y=0) → (x=0)
  - px: 0 → px: 1
  - py: 1 → py: 0
  - pz: 0 → pz: 1
The BIP framework – Flip-flop: Event Triggered model

\[ x=0, p_x, y=0, p_y, z=1, \tau_z := 0 \]
\[ y=1, p_y, x=0, p_x, z=1, \tau_z := 0 \]
\[ z=0, z := 1 \]
\[ z=1, \tau, z := 0 \]
Flip-flop: Event Triggered model

\[ \begin{align*}
px,py,pz & \quad \text{px,py,pz} \\
y=1 \land z=1, \tau & \quad \text{y=1} \land z=1, \tau \\
z:=0 & \quad z:=0 \\
\tau & \quad \tau \\
z:=1 & \quad z:=1 \\
x=0, \tau & \quad x=0, \tau \\
y=0, \tau & \quad y=0, \tau \\
x=1, \tau & \quad x=1, \tau \\
y=1, \tau & \quad y=1, \tau \\
px,py,pz & \quad px,py,pz \\
px,py,pz & \quad px,py,pz \\
px,py,pz & \quad px,py,pz \\
px,py,pz & \quad px,py,pz \\
px,py,pz & \quad px,py,pz \\
px,py,pz & \quad px,py,pz \\
px,py,pz & \quad px,py,pz \\
\end{align*} \]
Flip-flop: Data Triggered model

\[ z = \neg x \land \neg y \]

\[ x \lor y \]

\[ z := 0 \]

\[ x \land y \]

\[ z := 1 \]

\[ px, py, pz \]

\[ px1 \]

\[ py1 \]

\[ pz1 \]

\[ px2 \]

\[ py2 \]

\[ pz2 \]

\[ y1 := z1 \]

\[ y2 := z1 \]

\[ PR: px, py, pz \backslash \tau \]
Synchronous components

PR: syn ∖ all_other_ports

Micro-step
Synchronous mod2 counter

- **Zero'**
  - $g_{\text{flip}}$: $X=1$
  - $f_{\text{flip}}$: $Y:=0$
- **Zero**
- **One**
  - $g_{\text{flip}}$: $X=1$
  - $f_{\text{flip}}$: $Y:=1$
- **One'**

Modulo-2 counter
Synchronous mod\textsubscript{8} counter

\begin{align*}
\text{CN: } & \text{tick} = \{\text{tick}_0, \text{tick}_1, \text{tick}_2\}, \\ & f_{\text{tick}}: X_1 := Y_0; \quad X_2 := Y_1 \land Y_0
\end{align*}

\begin{align*}
\text{PR: } & \text{tick} \langle \text{flip}_0, \text{tick} \rangle \langle \text{flip}_1, \text{tick} \rangle \langle \text{flip}_2
\end{align*}
The BIP framework - traffic light for tramway crossing
Overview – Part 1

• Modeling real-time systems
  – The problem
  – Heterogeneity
  – Component-based construction

• Interaction modeling
  – Definition
  – Composition
  – Deadlock-freedom preservation

• Priority modeling
  – Definition
  – Composition

• The BIP framework
  – Implementation
  – Timed components
  – Event triggered vs. Data triggered

• Discussion
Discussion - Summary

• Framework for component-based modeling encompassing heterogeneity and relying on a **minimal set of constructs and principles**

• Clear separation between behavior and architecture
  - Architecture = interaction + priority
  - Correctness-by-construction techniques for deadlock-freedom and liveness, based on sufficient conditions on architecture (mainly)

• Other applications at Verimag
  - IF toolset allows layered description of timed systems
  - Methodology and tool support for generating scheduled code for real-time applications (work by S. Yovine et al.)
A component is defined as a point in the space:
Behavior × Interaction × Priority

Classes of components can be obtained by application of simple transformations

- Behavior: Decoupling interaction and computation; Loosening synchronization
- Interaction models: Fusing or merging connectors
- Priorities: adding/removing priority rules

Basis for property preservation results and correctness by construction
Discussion – expressiveness

Study Component Algebras \( CA = (B, GL, \oplus, \cong) \)
- \((GL, \oplus)\) is a monoid and \(\oplus\) is idempotent
- \(\cong\) is a congruence compatible with operational semantics

- Study classes of glue operators
- Focus on properties relating \(\oplus\) to \(\cong\)

Study notions of **expressiveness** characterizing structure
Given \( CA_i = (B_i, GL_i, \oplus_i, \cong_i), \ i=1,2, \)

\( CA_1 \) is more expressive than \( CA_2 \) if \( \forall P \)
\( \exists gl_2 \in GL_2 \) \( gl_2(B_1, \ldots, B_n) \) sat \( P \) \( \Rightarrow \exists gl_1 \in GL_1 \) \( gl_1(B_1, \ldots, B_n) \) sat \( P \)
**Discussion – expressiveness**

<table>
<thead>
<tr>
<th>PR</th>
<th>IM</th>
<th>$C \Rightarrow a \langle b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Problem: For given B, IM and PR which coordination problems can be solved?

Looking for a notion of expressiveness different from existing ones which

- Either completely ignore structure
- or use operators where separation between structure and behavior seems problematic e.g. hiding, restriction
Papers available at:
http://www-verimag.imag.fr/~sifakis/

• “A Framework for Component-based Construction”, SEFM05
  Keynote talk, September 7-9, 2005, Koblenz, pp 293-300.

• “Composition for Component-Based Modeling”, Science of Computer

• “Scheduler modeling based on the controller synthesis paradigm”

• “Component-based construction of deadlock-free systems”,
  FSTTCS03, LNCS 2194.

• “Priority Systems” Proceedings of FMCO’03, LNCS 3188
Overview – Part 2

- Timed systems
  - Definition
  - Examples

- Scheduler modeling
  - The role of schedulers
  - Control invariants
  - Scheduler specifications
  - Composability results

- Timed systems with priorities
  - Definition
  - Composition of priorities
  - Correctness-by-construction results

- The IF toolset
Timed systems – from untimed to timed systems

Methodology:

• Avoid over-specification which may lead to inconsistency

• Make explicit all the consequences of the constraints on interactions

• Define $\parallel_T$ so as to preserve properties such as well-timedness, and deadlock-freedom
Example: Vending Machine

Vending Machine

- insert → Vending Machine
- push → Vending Machine
- coin_back → coke
- coke
- x:=0
- x<20
- x=10
- true
- x<10

States:
- true
- x<20
- x<10

Transitions:
- insert: x:=0, x<20
- push: x<20, x:=0
- coin_back: x:=0, x<10
Example: temperature control system

Requ: Θ respects the bounds and rods can be reused after T
Timed systems – untimed systems: definition

Untimed system: A set of transitions

\[
\begin{array}{c}
\circlearrowright \\
\text{s} \\
\xrightarrow{\text{a, g, f}} \\
\text{s'} \\
\end{array}
\]

where

- \(S\) is a finite set of control states
- \(A\) is a set of actions
- \(\rightarrow \subseteq S \times A \times S\), a transition relation
- \(X\) a set of variables

Each transition is labeled with a guard and a transfer function

Operational semantics: A set of transitions

\[(s, x) \xrightarrow{\text{a}} (s', f(x))\]

where \(x\) is a valuation of \(X\) such that \(g(x) = \text{true}\)
Timed System - definition

Timed system: A set of transitions

\[ \phi_s \quad a, g, u, f \quad \phi_{s'} \]

where

- **u is an urgency condition** such that \( u \Rightarrow g \)
- Each control state \( s \) is labeled with a function \( \phi_s \) such that
  \( \phi_s(x,t) \) is the valuation of state variables when time progresses by \( t \) from state \((s,x)\).

Informal semantics:
  - Discrete transitions as for untimed systems
  - Notion of time progress: time can progress at \( s \) only if the urgency conditions of the transitions issued from \( s \) are false
A periodic process of period $T > 0$ and execution time $E$, $(E \leq T)$.

$t' = x' = 1$ at all states
Timed Systems - definition

A state is a pair \((s, x)\) where \(x\) is a valuation of \(X\)

**Discrete Transitions**
\[(s, x) - a_i \rightarrow (s_i, f_i(x)/x) \quad \text{if} \quad g_i(x) = \text{true}\]

**Time steps**
\[(s, x) - t \rightarrow (s, \phi_s(x,t)) \quad \text{if} \quad \forall t' < t \quad t_p(x+t') \quad \text{where} \quad t_p = \neg (\forall_i u_i)\]

*Time can progress as long as no urgency condition is true*
Timed Systems - relating urgency and time progress

\[ 3 \leq x \leq 5 \quad x=5 \quad 4<y \leq 7 \]

\[ \text{tp} = x \neq 5 \land (y \leq 4 \lor y > 7) \]

3≤ x≤5  
3≤ x≤5  

4<y ≤ 7  
4<y ≤ 7  

a  
a  

b  
b
Timed Systems – urgency types

\[ b = (a, g, u, f) \]

\[ g : a \text{ may be executed} \quad u : a \text{ must be executed} \]

\[ u \Rightarrow g \]

**Invariant:** If \( a \) cannot be executed then time can progress at \( s \)

\[ g \]

\[ u = g \]

**eager** (\( \epsilon \))

\[ u \downarrow g \]

**delayable** (\( \delta \))

\[ u = \text{false} \]

**lazy** (\( \lambda \))
Timed Systems: Urgency types

Replace urgency conditions by *urgency types* preserved by restriction of guards

\[ g^\lambda : \text{lazy guard } (u=false) \]

\[ g^\varepsilon : \text{eager guard } (u=g) \]

\[ g^\delta : \text{delayable guard } (u=g\downarrow) \]

Any TS can be transformed into an equivalent one with urgency types
Timed Systems - a periodic process

A periodic process of period $T > 0$ and execution time $E$, $(E \leq T)$.

$$t := 0 \quad (t = T) \epsilon$$

$$x := 0 \quad (t \leq T - E) \delta$$

$t' = x' = 1$ for all states
Timed Systems as transition systems

Q: set of states

\[ \rightarrow \subseteq Q \times A \times Q \quad q -a \rightarrow q' \quad \text{untimed transition} \]

\[ \rightarrow \subseteq Q \times R_+ \times Q \quad q -t \rightarrow q' \quad \text{time step} \]

Property (time additivity)

\[ q_1 -t_1 \rightarrow q_2 \quad \text{and} \quad q_2 -t_2 \rightarrow q_3 \quad \text{implies} \quad q_1 - (t_1 + t_2) \rightarrow q_3 \]

A run is a maximal sequence of transitions from states

\[ q_0 \quad q_1 \ldots \quad q_i \ldots \quad \text{such that} \quad q_i - t_i \rightarrow q_{i+1} \quad \text{or} \quad q_i - a_i \rightarrow q_{i+1} \]

\[ \text{time } [q_0, q_i] = \sum_{k \leq i} t_k \]

\[ q_0 \quad q_1 \ldots \quad q_i \ldots \quad \text{is time divergent if} \quad \forall k \in \mathbb{N} \ \exists i \quad \text{time } [q_0, q_i] > k \]

Important: Well-timed systems (only time divergent runs !)
Timed systems as transition systems - discrete vs. continuous

a TIMEOUT[2]b : execute a within 2 otherwise execute b

time unit 1

time unit 0.5

dense time
Timed systems as transition systems - discrete vs. continuous

\[ a \ (bc \ \text{TIMEOUT}[1] \ \text{AL2}) \ \text{TIMEOUT}[1] \ \text{AL1} \] for time unit 1

Possible abc within 0
Timed systems as transition systems - discrete vs. continuous

\[ a \ (bc \ \text{TIMEOUT}[1] \ AL2) \ \text{TIMEOUT}[1] \ AL1 \] for time unit 0.25

possible abc within 1.75
Timed systems as Transition Systems - discrete vs. continuous

\[ a \ (bc \ \text{TIMEOUT}[1] \ \text{AL2}) \ \text{TIMEOUT}[1] \ \text{AL1} \quad \text{for dense time} \]

possible abc within <2
Overview – Part 2

• Timed systems
  – Definition
  – Examples

• Scheduler modeling
  – The role of schedulers
  – Control invariants
  – Scheduler specifications
  – Composability results

• Timed systems with priorities
  – Definition
  – Composition of priorities
  – Correctness-by-construction results

• The IF toolset
Scheduler modeling - the role of schedulers

A scheduler is a **controller** restricting access to resources by triggering **controllable** interactions so as to respect timing constraints (state predicates) $K_0 = K_{\text{SCH}} \land K_{\text{POL}}$

- $K_{\text{SCH}}$ scheduling constraints (timing constraints on processes)
- $K_{\text{POL}}$ scheduling policy

```
Scheduler for $K_{\text{SCH}} \land K_{\text{POL}}$

controllable interaction

state

Interactions

Processes

Timed Model
```

QoS requirem
Scheduler modeling - example

A periodic process of period $T$ and completion time $E$

Actions
- $a$: arrive
- $b$: begin
- $f$: finish
- $p$: preempt
- $r$: resume

$t' = x' = 1$ at all states except stop ($x' = 0$)
Scheduler modeling - control invariants

A control invariant $K \implies K_0$

- Control invariants are preserved by uncontrollable actions
- It is possible to maintain the system in $K$ by executing controllable actions
Scheduler modeling - restriction by a constraint

The restriction of $TS$ by a constraint $K$ is a timed system $TS/K$

In $TS/K$, $K$ holds right before and right after the execution of any controllable action.

If $K$ is a control invariant of $TS$ then $TS/K$, is the scheduled (controlled) system.
Scheduler modeling – controller synthesis

There exists a scheduler maintaining $K_0$ if there exists a non empty control invariant $K$, $K \Rightarrow K_0$

For given $K_0$, the maximal control invariant $K$, $K \Rightarrow K_0$ can be computed as the result of a synthesis semi-algorithm $\text{SYNTH}(TS,K_0) = \lim_{i \to \infty} \{K_i\}$ where

$$K_{i+1} = K_i \land \text{contr-pre}(K_i)$$

from $K_0$

All states from which TS can be led to $K_i$ no matter how the environment behaves

$$\text{contr-pre}(K_i)$$
Scheduler modeling - invariants vs. control invariants

Def: $K$ is an invariant of $TS$ if it is preserved by the transition relation ($TS$ sat $inv(K)$)

- Any invariant is a control invariant
- $K$ is a control invariant of $TS$ if $K$ is an invariant of $TS/K$, that is $TS/K$ sat $inv(K)$
- $TS^{u}$ sat $inv(K)$ implies $TS/K$ sat $inv(K)$
Scheduler modeling – composability of control invariants

- Are control invariants preserved by conjunction?
- Is it possible to apply a composition principle by computing control invariants?

**Def:** A control invariant $K_1$ of $TS$ is **composable** if for all constraints $K_2$, $K_1$ is a control invariant of $TS/K_2$

- If $K_1$ is composable and $K_2$ is a control invariant of $TS/K_1$ then $TS/(K_1 \land K_2)$ sat inv $(K_1 \land K_2)$
- $K$ is composable iff $TS^u$ sat inv$(K)$
Scheduler modeling – composability of control invariants

$K_{\text{mutex}} = \neg (e_1 \land e_2)$ is a composable control invariant of $TS_1 \cup TS_2$
Scheduler modeling – composability of control invariants

\( TS_1 \cup TS_2 / K_{\text{mutex}} \)

\( K_{df} = K_{df1} \land K_{df2} \) is a control invariant of \( TS_1 \cup TS_2 \)

\( K_{df} \) is not a control invariant of \( TS_1 \cup TS_2 / K_{\text{mutex}} \)
Scheduler modeling – the scheduling constraint $K_{SCH}$

The scheduling constraint $K_{SCH}$ relates timing constraints of 3 different kinds

- *from the execution platform* e.g. execution times, latency times

- *from the external environment* about arrival times of triggering events e.g. periodic tasks

- *user requirements* e.g. QoS, which are timing constraints relating events of the real-time system and events of its environment e.g. deadlines, jitter
Scheduler modeling – the scheduling constraint $K_{SCH}$

Each shared resource induces a partition \{Sleep, Wait, Use\}.

- **Sleep**
  - arrive
  - $t:=0$
  - $T_{min} \leq t \leq T_{max}$

- **Wait**
  - begin
  - $x:=0$
  - $t \leq D - E_{max}$

- **Use**
  - $t \leq D$
  - finish
  - $E_{min} \leq x \leq E_{max}$
  - $t \leq D$

**Arrival time (t)**

**Execution time (x)**

**Deadline D**
Scheduler modeling – the scheduling constraint $K_{SCH}$

$$K_{SCH} = \bigwedge_i K_i^{SCH}$$

where $K_i^{SCH}$ expresses the property that no timing constraint is violated in process $i$.

For timelock-free process models with bounded guards, schedulability boils down to deadlock-freedom of processes.

$$K_{SCH} = s \land (t \leq T) \lor w \land (t \leq T-E) \lor u \land (x \leq E)$$
Scheduler modeling – the scheduling policy $K_{POL}$

$K_{POL}$ is the conjunction of scheduling policies for the set $R$ of shared resources

$$K_{POL} = \bigwedge_{r \in R} K_{r POL}$$

where

$$K_{r POL} = K_{r CONF} \land K_{r ADM}$$

- $K_{r CONF}$ says how conflicts for the acquisition of resource $r$ are resolved e.g. EDF, RMS, LLF

- $K_{r ADM}$ says which requests for $r$ are considered by the scheduler at a state e.g. masking
Scheduler modeling – the scheduling policy $K_{POL}$

$K_{POL}$: scheduling policy

$K_{ADM}$: admission control

$K_{CONF}$: Conflict resolution

$r^1 \ K^1_{ADM}$

$r^i \ K^i_{ADM}$

$r^n \ K^n_{ADM}$

$r^1 \ K^1_{CONF}$

$r^i \ K^i_{CONF}$

$r^n \ K^n_{CONF}$
Scheduler modeling – the scheduling policy $K_{POL}$: example

$K_{POL}$ for the Priority Ceiling Protocol

Admission control: “Process $P$ is eligible for resource $r$ if the current priority of $P$ is higher than the ceiling priority of any resource allocated to a process other than $P$”

Conflict resolution: “The CPU is allocated to the process with the highest current priority”
Scheduler modeling – composability results

• Any constraint $K_{pol}$ is a composable control invariant that is, 
  \[ \text{SYNTH}(TS, K_{pol}) = TS / K_{pol} \]

• Decomposition of the global synthesis problem 
  \[ \text{SYNTH}(TS, K_{sched} \land K_{pol}) = \text{SYNTH}(TS/K_{pol}, K_{sched}) \]

• Reduction to verification of $\text{SYNTH}(TS, K_{sched})$
  1. Choose a scheduling policy $K_{pol}$ such that the conflicts on controllable actions of $TS/K_{pol}$ are resolved
  2. Check $TS/K_{pol}$ sat $\text{inv}(K_{sched})$
Composability results - application

A scheduler design methodology supported by the Prometheus tool connected to Kronos

\[
K := K_{\text{sched}}; \\
\text{while} \quad \neg (\text{TS}/K \text{ sat inv}(K)) \quad \text{do} \\
\quad \text{choose } K_{\text{pol}}; \quad K := K_{\text{sched}} \land K_{\text{pol}} \\
\text{od}
\]
Overview – Part 2

- Timed systems
  - Definition
  - Examples

- Scheduler modeling
  - The role of schedulers
  - Control invariants
  - Scheduler specifications
  - Composability results

- Timed systems with priorities
  - Definition
  - Composition of priorities
  - Correctness-by-construction results

- The IF toolset
Timed Systems with priorities – about priorities

- Priorities are a special kind of restriction used to resolve conflicts between actions

- Priorities are commonly used in systems for resource management and scheduling

- Their combination with behavior raises some problems e.g. associativity of composition

- Have often been considered as “low” level concept e.g. “What It Means for a Concurrent Program to Satisfy a Specification: Why No One Has Specified Priority” Leslie Lamport, POPL, 1984
Timed Systems with priorities

Priority | Strengthened guard
---|---
\(a_1 \langle_0 a_2\) | \(g_1' = g_1 \land \neg g_2\)
\(a_1 \langle_5 a_2\) | \(g_1' = g_1 \land \neg \langle 5 \rangle g_2\)
\(a_1 \langle_\infty a_2\) | \(g_1' = g_1 \land \neg \langle \infty \rangle g_2\)

Notation: \(\langle k \rangle g(X) = \exists t \leq k \ g(X+t)\) (= eventually \(g\) within time \(k\))
Timed Systems with priorities

\( a_1 \prec_k a_2 \) means that \( a_1 \) is disabled when \( a_2 \) will be enabled within time \( k \)

Def: A priority order is a set of partial orders \( \prec = \{ \prec_k \mid \text{partial order on } A \} \) \( k \in \mathbb{R}^+ \) s.t.

\[
\begin{align*}
a_1 \prec_k a_2 \land a_2 \prec_m a_3 & \Rightarrow a_1 \prec_{k+m} a_3 \quad \text{(transitivity)}
\end{align*}
\]

Application of a priority order \( \prec \)

\[
\begin{align*}
g_i' &= g_i \land \left( \bigwedge_{ai} \prec_{k} \bigwedge_{am} \prec_{k} g_m \right)
\end{align*}
\]
A *timed system with priorities* is a pair \((TS, pr)\) where \(pr\) is a set of *priority rules* \(pr = \{C_i, \langle i \rangle_i\}\) with

- \(\{C_i\}_i\) is a set of disjoint time invariant predicates
- \(\{\langle i \rangle_i\}_i\) is a set of priority orders

\[pr = \{ C_i \rightarrow \langle i \rangle_i \}_i\]

\[g_i' = g_i \wedge \bigwedge C \rightarrow \langle \in pr \ (C \Rightarrow \bigwedge_{\langle k \rangle} \bigwedge_{ai} <k> g_m)\]

**Activity Preservation Theorem:** \(\Diamond \bigvee_i g_i = \Diamond \bigvee_i g_i'\)
If $K$ is a constraint characterizing a set of deadlock-free states of $TS$ then there exists a set of priority rules $pr$ such that $(TS, pr)$ preserves $K$.

For any control invariant $K$ of $TS$ there exists a set of dynamic priority rules $pr$ such that the scheduled system $TS/K = (TS, pr)$.

Any feasible scheduling policy $K_{POL}$ induces a restriction that can be described by priorities.
Timed Systems with priorities - fixed priority policy

\[ w_1 \land w_2 \rightarrow b_1 \prec_k b_2 \text{ for some } k \]
Timed Systems with priorities - FIFO policy

t₁ ≤ t₂ → b₁ <₀ b₂     t₂ ≤ t₁ → b₂ <₀ b₁

s₁
\( a₁ \)
\( t₁ = T₁ \)
\( t₁ := 0 \)

w₁
\( b₁ \)
\( t₁ ≤ T₁ - E₁ \)
\( x₁ := 0 \)

x₁ = E₁
f₁

s₂
\( a₂ \)
\( t₂ = T₂ \)
\( t₂ := 0 \)

w₂
\( b₂ \)
\( t₂ ≤ T₂ - E₂ \)
\( x₂ := 0 \)

e₂
f₂

x₂ = E₂
Timed Systems with priorities - LLF policy

\[ L_1 \leq L_2 \rightarrow b_2 \ll_0 b_1 \quad \text{L}_2 \leq L_1 \rightarrow b_1 \ll_0 b_2 \]

where \( L_i = T_i - E_i - t_i \),

\[
\begin{align*}
&x_1 = E_1 \\
&t_1 = T_1 \\
&t_1 := 0 \\
&f_1 \\
&w_1 \\
&b_1 \\
&t_1 \leq T_1 - E_1 \\
&x_1 := 0 \\
&s_1 \\

&x_2 = E_2 \\
&t_2 = T_2 \\
&t_2 := 0 \\
&f_2 \\
&w_2 \\
&b_2 \\
&t_2 \leq T_2 - E_2 \\
&x_2 := 0 \\
&s_2
\end{align*}
\]
Def: If $\langle \mathbf{1}, \mathbf{2} \rangle$ are two priority orders on $A$ then $\langle \mathbf{1} \oplus \mathbf{2} \rangle$ is the least priority order (if it exists) s.t. $\langle \mathbf{1} \cup \mathbf{2} \rangle \subseteq \langle \mathbf{1} \oplus \mathbf{2} \rangle$

• Note: $\langle \mathbf{1} \oplus \mathbf{2} \rangle$ is the closure of $\langle \mathbf{1} \cup \mathbf{2} \rangle$ by using the transitivity rule

• We extend the operation $\oplus$ to priority rules $\mathbf{pr}_i$

$$\forall q \in Q. \ (\mathbf{pr}_1 \oplus \mathbf{pr}_2)(q) = \mathbf{pr}_1(q) \oplus \mathbf{pr}_2(q)$$
Timed Systems with priorities - composition of priorities

Results:
• The operation $\oplus$ is partial, associative and commutative
• $\text{pr}_1(\text{pr}_2(B)) \neq \text{pr}_1(\text{pr}_2(B))$
• $\text{pr}_1(\text{pr}_2(B)) = \text{pr}_1(\text{pr}_2(B))$ if $\text{pr}_1 \oplus \text{pr}_2 = \text{pr}_1 \cup \text{pr}_2$
• Priorities preserve deadlock-freedom

We take by definition

\[
\begin{array}{c}
\text{pr}_2 \\
\text{pr}_1 \\
\hline
. . \\
\hline
\end{array}
\quad = \quad
\begin{array}{c}
\text{pr}_1 \oplus \text{pr}_2 \\
\hline
. . \\
\hline
\end{array}
\]
Timed Systems with priorities – mutual exclusion

Idea: Give infinitely higher priority to the process using the resource

\[ w_1 \land e_2 \rightarrow b_1 \langle \infty \rangle f_2 \quad w_2 \land e_1 \rightarrow b_2 \langle \infty \rangle f_1 \]
Timed Systems with priorities – mutual exclusion

The behavior after application of mutual exclusion constraints
Timed Systems with priorities – mutual exclusion

Risk of deadlock: The composition is not a priority order!
Timed Systems with priorities – mutual exclusion + FIFO policy

\[ t_1 \leq t_2 \rightarrow b_1 \preceq_0 b_2 \quad t_2 \leq t_1 \rightarrow b_2 \preceq_0 b_1 \]

\[ w_1 \land e_2 \rightarrow b_1 \preceq_\infty f_2 \quad w_2 \land e_1 \rightarrow b_2 \preceq_\infty f_1 \]
Overview – Part 2

- Timed systems
  - Definition
  - Examples

- Scheduler modeling
  - The role of schedulers
  - Control invariants
  - Scheduler specifications
  - Composability results

- Timed systems with priorities
  - Definition
  - Composition of priorities
  - Correctness-by-construction results

- The IF toolset
Timed Systems with priorities – liveness

Run: a maximal sequence of successive transitions in a TS

\[ q_0 \rightarrow t_0 \rightarrow q_0' \rightarrow a_1 \rightarrow q_1 \rightarrow t_1 \rightarrow q_1' \rightarrow a_2 \rightarrow \ldots \ldots \]

\[ q_i \rightarrow t_i \rightarrow q_i' \rightarrow a_i \rightarrow q_{i+1} \rightarrow t_{i+1} \rightarrow \ldots \ldots \]

Timelock: a run where the total time elapsed is bounded

Livelock: a run where only a finite number of transitions occur

\[ \text{LIVE} = \text{Timelock-free} + \text{Livelock-free} \]
Timed Systems with priorities – structural liveness

Enforce liveness satisfaction by appropriate structural restrictions preserved by composition operators.

2 structural properties easy to check:

- structurally non-Zeno
- locally livelock-free

- timelock-free
- livelock-free

- structurally live
Timed Systems with priorities – structural liveness

**Structurally non-Zeno:** any circuit of the control graph has some clock reset and tested against some positive lower bound

**Locally Livelock-free:** if time can progress then some action will be executed

\[ \text{in}(s) \Rightarrow \Diamond \lor_i u_i \]

\[ \text{SnZ} \Rightarrow \text{TLF} \quad \text{LLLF} \Rightarrow \text{LLF} \]

Structurally live = SnZ + LLLLF
Timed Systems with priorities – structural liveness

A periodic process of period $T>0$ and execution time $E$, $(E \leq T)$.

This process is structurally live:

• Timelock-free because $SnZ$

• Locally LLF because
  
  \[
  \text{in(wait)}=(t=0) \Rightarrow \Diamond (t=T-E) = t \leq T-E \\
  \text{in(exec)}=(x=0) \Rightarrow \Diamond (x=E) = x \leq E \\
  \text{in(sleep)}=(x=E) \Rightarrow \Diamond (t=T) = t \leq T
  \]

???
Timed Systems with priorities – structural liveness

A periodic process of period $T>0$ and execution time $E$, $(E \leq T)$.

This process is structurally live:

• Timelock-free because $SnZ$

• Locally LLF because

  $\text{in(wait)}=(t=0) \Rightarrow \Diamond (t=T-E) = t \leq T-E$

  $\text{in(exec)}=(x=0) \land (t \leq T-E) \Rightarrow \Diamond (x=E) \land (t \leq T)$

  $\text{in(sleep)}=(x=E) \land (t \leq T) \Rightarrow \Diamond (t=T) = t \leq T$
Timed Systems with priorities – structural liveness

Theorem:
Priorities preserve the 3 structural properties, thus they preserve structural liveness that is if TS is structurally live then (TS, pr) is structurally live too
Flexible Composition - Untimed systems

Preserves deadlock-freedom of untimed components
Flexible Composition - timed systems

For $bi=(ai, gi, ui, ri)$, take

$$b1|b2 = (a1|a2, g1|g2, u1|u2, r1\cup r2)$$

where

$g1|g2$ is a monotonic function (synchronization mode)

$u1|u2 = (g1|g2)\land(u1\lor u2)$

PROPERTIES: Maximal progress+Activity preservation
Flexible Composition: Composition of Guards

\[ (s_1, s_2) \]
\[ (s_1', s_2') \]

\[ a_1 \]
\[ g_1 \]
\[ \text{等待} \]
\[ g_1 \land g_2 \]
\[ \text{min} \]
\[ g_1 \text{ min } g_2 = g_1 \land \langle \rangle g_2 \lor g_2 \land \langle \rangle g_1 \]
\[ \text{anticipation} \]
\[ g_1 \text{ or } g_2 = g_1 \lor g_2 \]

\[ g_1 \text{ and } g_2 = g_1 \land g_2 \]
\[ g_1 \text{ max } g_2 = g_1 \land \langle \rangle g_2 \lor g_2 \land \langle \rangle g_1 \]
\[ \text{waiting} \]
\[ g_1 \text{ or } g_2 = g_1 \lor g_2 \]

Notation: \( \langle \rangle g(X) = \exists t \ 0 \leq t \ g(X-t) \) (once \( g \))
Composition of Guards: and-Synchronization

Example:
\( g_1 = 2 \leq x \leq 3 \)
\( g_2 = 1 \leq y \leq 2 \)
\( g_1 \text{ and } g_2 = g_1 \land g_2 \)

\( g_1' = g_1 \land \neg \Box (g_1 \land g_2) = (2 \leq x \leq 3) \land (y > 2 \lor x - y > 2 \land y > 0) \)

\( g_2' = g_2 \land \neg \Box (g_1 \land g_2) = (1 \leq y \leq 2) \land (x > 3 \lor y - x > 0 \land x > 0) \)
Composition of Guards: max-Synchronization

Example:

\[ g_1 \text{max} g_2 = (2 \leq x \leq 3) \land 1 \leq y \lor 2 \leq x \land (1 \leq y \leq 2) \]

\[ g_1' = g_1 \land \neg \Diamond (g_1 \text{max} g_2) = \text{false} \quad g_2' = g_2 \land \neg \Diamond (g_1 \text{max} g_2) = \text{false} \]
Flexible Composition: Producer-Consumer

\[
\begin{align*}
2 & \leq x \leq 5 \\
x & := 0 \\
1 & \leq x \leq 3 \\
x & := 0 \\
3 & \leq y \leq 6 \\
y & := 0 \\
2 & \leq y \leq 4 \\
y & := 0 \\
3 & \leq y \leq 6 \\
y & := 0 \\
2 & \leq x \leq 5 \\
x & := 0 \\
3 & \leq y \leq 6 \\
y & := 0 \\
2 & \leq x \leq 5 \\
x & := 0 \\
3 & \leq y \leq 6 \\
y & := 0 \\
and: \\ g & = 1 \leq x \leq 3 \land 2 \leq y \leq 4 \\
\text{deadlock in most cases} \\
\text{absence of deadloc}
\end{align*}
\]
Composition of Guards: min-Synchronization

Example:

$g_1^{\text{ming}2} = (2 \leq x \leq 3) \land y \leq 2 \lor x \leq 3 \land (1 \leq y \leq 2)$

$g_1' = g_1 \land \neg \square (g_1^{\text{ming}2}) = (2 \leq x \leq 3) \land y > 2$

$g_2' = g_2 \land \neg \square (g_1^{\text{ming}2}) = x > 3 \land (1 \leq y \leq 2)$
Composition of Typed Guards

For $\tau, \tau_1, \tau_2, \tau_3 \in \{\varepsilon, \lambda\}$, $\mid \in \{\text{and, max, min, or}\}$

\[
g_1 \mid (g_2 \mid g_3) = (g_1 \mid g_2) \mid (g_1 \mid g_3)
\]

\[
g_1 \tau \mid g_2 \tau = (g_1 \mid g_2) \tau
\]

\[
g_1 \varepsilon \mid g_2 \lambda = g_1 \varepsilon \mid (g_2 \mid \neg g_1)
\]

\[
g_1 \varepsilon \mid \max g_2 \lambda = (g_1 \mid \neg g_2) \varepsilon \mid (g_2 \mid \neg g_1) \lambda
\]

\[
g_1 \varepsilon \mid \min g_2 \lambda = (g_1 \mid g_2) \varepsilon \mid (g_2 \mid g_1) \lambda
\]

\[
g_1 \delta \mid \delta = (g_1 \mid g_2)
\]

\[
g_1 \delta \mid \max g_2 \delta = (g_1 \mid \neg g_2) \delta \mid (g_2 \mid \neg g_1) \delta
\]

\[
g_1 \delta \mid \min g_2 \delta = (g_1 \mid g_2) \delta \mid (g_2 \mid g_1) \delta
\]
MAX, MIN: powerful synchronization primitives

\[ x_1 := 0 \quad x_2 := 0 \]

\[ x_1 := 0 \quad x_2 := 0 \]

\[ g_{12} = 2 \leq x_1 \leq 4 \quad \land \quad 3 \leq x_2 \lor 2 \leq x_1 \land \quad 3 \leq x_2 \leq 5 \]

\[ x_1 := 0 \quad x_2 := 0 \]

\[ 2 \leq x_1 \leq 4 \quad \land \quad 3 \leq x_2 \lor 2 \leq x_1 \land \quad 3 \leq x_2 \leq 5 \]
Theorem: and-composition preserves

- Structural liveness if $\Diamond gi = \Diamond ui$
- Moreover, individual liveness if $\Diamond \neg gi$
Structural liveness preservation - best effort synchronization of go1, go2

\[ g_1 \land g_2 = (t_1 \leq T_1 - E_1) \land (t_2 \leq T_2 - E_2) \]

\[ g_1' = (t_1 \leq T_1 - E_1) \land (t_2 \geq T_2 - E_2) \]

\[ g_2' = (t_2 \leq T_2 - E_2) \land (t_1 \geq T_1 - E_1) \]
Application: Petri Nets with Deadlines

$g : \text{guard}$

$u : \text{urgency condition such that } u \Rightarrow g$

$r : \text{set of clocks to be reset}$

Firing rules

• A transition is enabled if it is enabled in the PN and the corresponding guard is true
• Time progress stops if the deadline of some transition is true
Timed Petri Nets

Token state: available, unavailable

Firing asap by available tokens

Unavailable to available within $[l_i, u_i]$
PN with Synchronization Modes

\( \text{mode} \in \{\text{and, max, min, or}\} \)
\( \tau \in \{\lambda, \delta, \epsilon\} \)

\[ \begin{align*}
\text{a1} \quad & \quad \text{a2} \\
p1 \quad & \quad p2 \\
g1 \quad & \quad g2 \\
p3 \quad & \quad p4 \\
\end{align*} \]

\[ \begin{align*}
x1 &= 0 \\
x2 &= 0 \\
l1 \leq x1 \leq u1 \\
l2 \leq x2 \leq u2 \\
\end{align*} \]
APPLICATION: specification of multimedia documents

Syntax of documents
\[
D ::= O_i \in O \mid D \ op \ D
\]
where \( op \in \{\text{MEETS, EQUAL, PARMIN, PARMAX}\} \)

- Each \( O_i \) has a duration interval \([l_i, u_i]\)

- Operators build a composite document by imposing constraints on the starting and finishing times of the components

![Diagram of D1 MEETS D2 and D1 PAR_OP D2]

SYNCHRONIZATION
APPLICATION: specification of multimedia documents

Oi \([l_i, u_i]\) \[\xrightarrow{\text{MEETS}}\] D1 \[\xrightarrow{\text{PAR\_OPD2}}\] D2

\[\xi := 0\] \(l_i \leq \xi \leq u_i\)

\[\xrightarrow{\text{PN1 \(g_1\) \rightarrow \text{PN2 \(g_2\)}}}\]

\[\xrightarrow{\text{PN1 \(g_1\) \leftrightarrow \text{PN2 \(g_2\)}}}\]

\[\xrightarrow{\text{MODE}}\]

\[\xrightarrow{\text{EQUALS \(\Rightarrow\) AND}}\]

\[\xrightarrow{\text{PARMIN \(\Rightarrow\) MIN}}\]

\[\xrightarrow{\text{PARMAX \(\Rightarrow\) MAX}}\]

188
APPLICATION: specification of multimedia documents

\(((\text{video} \text{PARMIN} \text{button}) \text{MEET} \text{image}) \text{EQUALS} (\text{applet} \text{PARMAX} \text{sound}))\)
Application: Specification of Multimedia Documents

\(((\text{video \text{PARMIN} \text{button}) \text{MEETS} \text{image}) \text{ EQUALS} (\text{applet \text{PARMAX} \text{sound}}))\)

\[
\begin{align*}
\text{applet}[20,30] & \quad 20+a \\
\text{sound}[5,60] & \quad 30+b \\
\text{video}[35,40] & \quad 10+b \\
\text{button}[10,\infty] & \quad \text{min} \\
\text{image}[10,20] & \quad 20 \\
\end{align*}
\]

\[
0 \leq a \leq 10 \quad 0 \leq b \leq 30
\]

A dynamic schedule for button and applet uncontrollable
Application: Specification of Multimedia Documents

x = y := 0

applet

sound

video

button

image

10 ≤ y ≤ 40

y := 0

20 ≤ x ≤ 60 ∧ 10 ≤ y ≤ 20

TS
Discussion: Two Approaches for Composition of TS

**STRICT (NON FLEXIBLE)**
- Preserves urgency - risk of deadlock
- Adequate for « responsive cooperation »
- Constraint- oriented

**FLEXIBLE (NON STRICT)**
- Relax urgency to avoid timelock
- Adequate for « asynchronous» cooperation
- Design/Synthesis oriented
Discussion : Flexible Composition

Timed System = Composition of timed actions
• Urgency constraints are associated with actions
• Possibility to guarantee time progress by construction
• Variety of extensions depending on the choice of waiting times
• Use of modalities: just a macro-notation in most cases

Parallel Composition
• Activity preservation + Maximal progress
• Powerful synchronization primitives - avoiding state explosion
• Modeling Timed Petri nets
Discussion: Correctness by Construction

Structural properties

• Easy to check on components
• Compositionality rules for priorities and flexible composition
• Establishing LLLF may require strengthening of guards
Overview – Part 2

- **Timed systems**
  - Definition
  - Examples

- **Scheduler modeling**
  - The role of schedulers
  - Control invariants
  - Scheduler specifications
  - Composability results

- **Timed systems with priorities**
  - Definition
  - Composition of priorities
  - Correctness-by-construction results

- **The IF toolset**
The IF toolset: objectives

Model-based development of real-time systems

Use of high level modeling and programming languages
• Expressivity for faithful and natural modeling
• Cover functional and extra-functional aspects
• Openness

Model-based validation
• Combine static analysis and model-based validation
• Integrate verification, testing, simulation and debugging

Applications:
Protocols, Embedded systems, Asynchronous circuits, Planning and scheduling
The IF toolset: approach

Modeling and programming languages (SDL, UML, SCADE, Java …)

IF: Intermediate Format, based on a general and powerful semantic model

Optimisation and abstraction

Transition systems

- simulation
- test
- verification1
- verification2
- verification3

state explosion
The IF toolset: challenges

Find an adequate intermediate representation

**Expressiveness:** direct mapping of concepts and primitives of high modeling and programming languages

- asynchronous, synchronous, timed execution
- buffered interaction, shared memory, method call …

Use information about structure for efficient validation and traceability

**Semantic tuning:** when translating languages to express semantic variation points, such as time semantics, execution and interaction modes
The IF toolset - IF notation: system description

Processes (Behavior)
- extended timed systems
  (non-determinism, dynamic creation)

Data
- predefined data types
  (basic types, arrays, records)
- abstract data types

Interactions
- asynchronous channels
- shared variables

Dynamic
priorities
The IF toolset: - IF notation: the basic model (ACTA)

x, y: var
t: timer

?g, a
A process instance:
- executes asynchronously with other instances
- can be dynamically created
- owns local data (public or private)
- owns a private FIFO buffer

Inter-process interactions:
- asynchronous signal exchanges (directly or via signalroutes)
- shared variables
The IF toolset - IF notation: system description

const N1 = ... ;  // constants
type t1 = ... ;    // types

signal s2(t1, t2),  // signals
  // signalroutes
signalroute sr1(1) ...  // route attributes from P1 to P3

// processes
process P1(N0)
  ...  // data +
  behaviour
endprocess;

... process P3(N3)
  ... endprocess;

...
The IF toolset -IF notation: process description

Process = hierarchical, timed system

process P1(N1);
  parameters
  fpar ... ;

  // types, variables, constants, procedures
  state s0 ... ;
    ... // transition t1
  endstate;

  state s1 #unstable ... ;
    ... // transitions t2, t3
  endstate;

  ... // states s2, s3, s4
endprocess;
The IF toolset - IF notation: dynamic creation

• process creation:

\[ p := \text{fork client (true)} \]

- pid of the newly created instance
- process name
- parameters

- a new instance is created

• process destruction:

\[ \text{kill client(2)} \]
\[ \text{kill p} \]

- kill expression
- pid expression

- the instance is destroyed, together with its buffer, and local data

• process termination:

\[ \text{stop} \]

- the “self” instance is destroyed, together with its buffer, and local data
The IF toolset - IF notation: transition description

**transition** = \( urgency + trigger + body \)

**state** \( s_0 \)

... \( urgency \)

\begin{align*}
\text{urgency} & \quad \text{eager} \\
& \quad \text{provided} \quad x! = 10; \\
& \quad \text{when} \quad c_2 \geq 4; \\
& \quad \text{input} \quad \text{update}(m); \\
& \quad \text{body} \quad \ldots \\
& \quad \text{nextstate} \quad s_1; \\
& \quad \ldots \\
& \quad \text{endstate}; \\
\end{align*}

... \( = \) \( trigger \)

\( = \text{untimed guard} \)

\( = \text{timed guard} \)

\( = \text{signal consumption from the process buffer} \)

\( = \text{statement list} \)

\( = \text{sequential, conditional, or iterative composition} \)

**statement** = data assignment

message emission,

process or signalroute creation or destruction, ...
The IF toolset- IF notation: data and types

Variables:
• are **statically typed** (but *explicit conversions* allowed)
• can be declared **public** (= shared)

Predefined basic types: integer, boolean, float, pid, *clock*

Predefined type constructors:
• (integer) interval:  **type** fileno = range 3..9;
• enumeration:  **type** status = enum open, close endenum;
• array:  **type** vector = array[12] of pid
• structure:  **type** file = record f fileno; s status endrecord;

Abstract Data Type definition facilities …
The IF toolset - IF notation: interactions

signal route = connector = process to process communication channel with **attributes**, can be **dynamically** created

**signalroute** s1(1) #unicast #lossy #fifo

from server to client with grant, fail;

attributes:

- queuing policy: **fifo** | **multiset**
- reliability: **reliable** | **lossy**
- delivery policy: **peer** | **unicast** | **multicast**
- delay policy: **urgent** | delay[l,u] | rate[l,u]
The IF toolset - IF notation: interactions (delivery policies)

- **Peer**
  - server(0) to client(1)

- **Unicast**
  - server(0) to client(0), client(1), client(2)

- **Multicast**
  - server(0) to client(0), client(1), client(2)

- to one specific instance
- to a randomly chosen instance
- to all instances
The IF toolset - IF notation: interactions (signal exchange)

Signal emission (non blocking):

- to a specific process: `output req (3, open) to server(2);`
- via a signalroute: `output req(3, open) via s0(1);`
- mixed: `output token via link(1) to client(k+1)%N;`

Signal consumption (blocking):

- `input req (f, s);`
const NS= ... , NC= ... ;
type file= ... , status= ... , reason= ... ;

signal stop(), req(file, status), fail(reason), grant(), abort(), update(data);

signalroute s0(1) #multicast
from server to client with abort;
signalroute s1(1) #unicast #lossy
from server to client with grant,fail;
signalroute s2(1) #unicast
from client to server with req;

process server(NS) ... endprocess;
process client(NC) ... endprocess;
The model of time [timed systems]

- global time → same clock speed in all processes
- time progress in stable states only → transitions are instantaneous
The IF toolset - IF notation: timed behavior

- **operations on clocks**
  - set to value
  - deactivate
  - read the value into a variable

- **timed guards**
  - comparison of a clock to an integer
  - comparison of a difference of two clocks to an integer

```
state send;
output sdt(self,m,b) to {receiver}0;
set t:= 10;

nextstate wait_ack;
endstate;

state wait_ack;
input ack(sender,c);
...

when 10 < t < 20 ;
...
endstate;
```
The IF toolset - IF notation: dynamic priorities

• priority order between process instances $p_1, p_2$ (free variables ranging over the active process set)

\[
\text{priority_rule_name} : p_1 < p_2 \text{ if condition}(p_1,p_2)
\]

• semantics: only maximal enabled processes can execute

• scheduling policies
  – fixed priority: $p_1 < p_2$ if $p_1$ instanceof T and $p_2$ instanceof R
  – run-to-completion: $p_1 < p_2$ if $p_2 = \text{manager}(0).\text{running}$
  – EDF: $p_1 < p_2$ if $\text{Task}(p_2).\text{timer} < \text{Task}(p_1).\text{timer}$ (p1)
IF toolset - overall architecture

**Objecteering**
- UML
  - aml2if

**Rational Rose**
- RT/UML
  - OMEGA
  - uml2if

**ObjectGeode**
- SDL
  - sdl2if

**IF Exploration Platform**
- IF Description
- IF Static Analyzer
- Test Generation
  - TGV
    - Test Suites
    - SPIDER
  - model construction
  - model checking
  - guided simulation
    - mincost path extraction
      - schedules

**Static Analyzer**
- LASH
- RMC
- TReX
- CADP
- SPIDER
- LTS
IF toolset - Core components

- syntactic transformation tools:
  - static analyser
  - code generator

- C/C++ code
  - predefined modules
    - (time, channels, etc.)

- interaction model
  - priorities (scheduling)

- state space representation

LTS exploration tools
  -- debugging
  -- model checking
  -- test generation
IF toolset - core components: exploration platform

active instances

process 1

I_1:P_1

I_2:P_1

process 2

I_1:P_2

I_2:P_2

process j

I_k:P_j

Time module

run

run

step

step

step

run

output

create

run

set, reset

run

Interaction model

execution control

priorities (scheduling)

Succ?

Succ!
IF toolset - core components: exploration platform (time)

**Dedicated module**
- including clock variables
- handling dynamic clock allocation (set, reset)
- checking timing constraints (timed guards)
- computing time progress conditions w.r.t. actual deadlines and
- fires timed transitions, if enabled

Two implementations for discrete and continuous time (others can be easily added)

**i) discrete time**
- clock valuations represented as varying size integer vectors
- time progress is explicit and computed w.r.t. the next enabled deadline

**ii) continuous time**
- clock valuations represented using varying size difference bound matrices (DBMs)
- time progress represented symbolically
- non-convex time zones may arise because of deadlines: they are represented implicitly as unions of DBMs
IF toolset - case studies: protocols

**SSCOP**
Service Specific Connection Oriented Protocol

**MASCARA**
Mobile Access Scheme based on Contention and Reservation for ATM
case study proposed in *VIRES ESPRIT LTR*

**PGM**
Pragmatic General Multicast
case study proposed in *ADVANCE IST-1999-29082*
IF toolset - ase studies: asynchronous circuits

timing analysis

functional validation
IF toolset - case studies: embedded software

Ariane 5 Flight Program

joint work with EADS Lauchers


K9 Rover Executive


IF toolset - Ariane-5 flight program
IF toolset - Ariane-5 flight program: the model

- built by reverse engineering by EADS-LV

- two independent views
  1. asynchronous
     - high level, non-deterministic, abstracts the whole program as communicating extended finite-state machines
  2. synchronous
     - low level, deterministic, focus on specific components ...

- we focus on the asynchronous view
IF toolset - Ariane-5 flight program: architecture

OBC (On Board Computer)

Regulation
engines/boosters
ignition/extinction

Configuration
stage/payload
separation

Control
Navigation
Guidance
Algorithms

Ground

OBC (Redundant)

~3500 lines
of SDL code
initiate sequences of “regulation” commands at right moments in time:
- at $T_0 + \Delta_1$ execute $\text{action}_1$
- at $T_0 + \Delta_2$ execute $\text{action}_2$
  ...  
- at $T_0 + \Delta_n$ execute $\text{action}_n$

if necessary, stopped at any moment

described as “sequential” processes, moving on specific, precise times
IF toolset - Ariane-5 flight program: configuration components

- initiates “configuration” changes depending on:
  - flight phase: ground, launch, orbit, …
  - control information: reception of some signal, …
  - time: eventually done in \([T_0+L,T_0+U]\)

- described as processes combining signal and timeout-driven transitions
the opening action eventually happens between $T_{early}$ and $T_{late}$ moments, if possible, on the reception on the open signal.
IF toolset - Ariane-5 flight program: control components

• compute the flight commands depending on the current flight evolution
  – guidance, navigation and control algorithms

• abstracted over-simplified processes
  – send flight commands with some temporal uncertainty
time non-deterministic:
the firing signal can be sent between $T_0 + L$ and $T_0 + U$

$\begin{cases} 
T_0 + L \leq \text{now} \\
\text{now} \leq T_0 + U 
\end{cases}$
output firing to vulcain


time deterministic:
the firing signal is sent exactly at $T_0 + K$

$T_0 + K = \text{now}$
output firing to vulcain

init

lazy

done

eager

init
done
• general requirements
  – e.g. no deadlock, no timelock

• overall system requirements
  – e.g. flight phase order
  – e.g. stop sequence order

• local component requirements
  – e.g. activation signals arrive eventually in some predefined time intervals
IF toolset - Ariane-5 flight program: validation (model exploration)

- test simple properties by random or guided simulation
- several inconsistencies because timing does not respect causality e.g., deadline missed because of $\Delta_1 > \Delta_2$

\[
\text{now} = T_0 + \Delta_1
\]

\[
\text{output status}
\]